# DW_ram_2r_w_a_dff

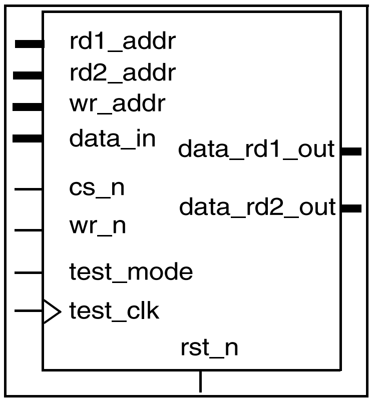
## Write-Port, Dual-Read-Port RAM (Flip-Flop-Based)

Version, STAR, and myDesignWare Subscriptions: IP Directory

## Features and Benefits

**Revision History**

- Parameterized word depth
- Parameterized data width
- Asynchronous static memory
- Parameterized reset implementation
- High testability using DFT Compiler



## Description

DW_ram_2r_w_a_dff implements a parameterized, asynchronous, three-port static RAM.

**Table 1-1 Pin Description**

| Pin Name | Width | Direction | Function |
|---|---|---|---|
| rst_n | 1 bit | Input | Reset, active low |
| cs_n | 1 bit | Input | Chip select, active low |
| wr_n | 1 bit | Input | Write enable, active low |
| test_mode | 1 bit | Input | Enables `test_clk` |
| test_clk | 1 bit | Input | Test clock to capture data during `test_mode` |
| rd1_addr | ceil($\log_2$[*depth*]) bits | Input | Read1 address bus |
| rd2_addr | ceil($\log_2$[*depth*]) bits | Input | Read2 address bus |
| wr_addr | ceil($\log_2$[*depth*]) bits | Input | Write address bus |
| data_in | *data_width* bits | Input | Input data bus |
| data_rd1_out | *data_width* bits | Output | Output data bus for read1 |
| data_rd2_out | *data_width* bits | Output | Output data bus for read2 |

**Table 1-2 Parameter Description**

| Parameter | Values | Description |
|---|---|---|
| data_width | 1 to 256 | Width of `data_in` and `data_out` buses. Default = none |
| depth | 2 to 256 | Number of words in the memory array (address width).<br>Default: None |
| rst_mode | 0 or 1<br>Default: 1 | Determines if the rst_n input is used.<br>■ 0: `rst_n` initializes the RAM<br>■ 1: `rst_n` is not connected |

**Table 1-3 Synthesis Implementations**

| Implementation Name | Function | License Feature Required |
|---|---|---|
| rtl[a] | Synthesis model | DesignWare |

a. The implementation, “rtl,” replaces the obsolete implementation, “str.” Existing designs that specify the obsolete implementation (“str”) will automatically have that implementation replaced by the new superseding implementation (“rtl”) as will be noted by an information message (SYNDB-36) generated during DC compilation.

**Table 1-4 Simulation Models**

| Model | Function |
|---|---|
| DW06.DW_RAM_2R_W_A_DFF_CFG_SIM | VHDL simulation configuration |
| dw/dw06/src/DW_ram_2r_w_a_dff_sim.vhd | VHDL simulation model source code |
| dw/sim_ver/DW_ram_2r_w_a_dff.v | Verilog simulation model source code |

The write data enters the RAM through the `data_in` input port, and is read out through either the `data_rd1_out` port or the `data_rd2_out` port. The RAM is constantly reading regardless of the state of `cs_n`.

The `rd1_addr`, `rd2_addr`, and `wr_addr` ports are used to address the *depth* words in memory. If `rd1_addr` or `rd2_addr` contains a value beyond the maximum depth, then that output port is driven low. For example, if `rd1_addr` = 7 hex and `depth` = 6, then the `data_rd1_out` bus is driven low. If `rd2_addr` is beyond the maximum depth, then `data_rd2_out` is driven low.

For `wr_addr` beyond the maximum depth, nothing occurs and the data is lost. No warnings are given during simulations when an address beyond the scope of *depth* is used.

**Attention** This component contains clock signals for internal flip-flops that are derived from the `wr_n` and `test_clk` ports. To keep hold times and internal clock skews to a minimum, you should consider instances of this component to be individual floorplanning elements.

## Chip Selection, Reading and Writing

The `cs_n` input is the chip select, active low signal that enables data to be written to the RAM. The RAM is constantly reading, regardless of the state of `cs_n`.

When `cs_n` is low and there is a low-to-high transition of the write enable, `wr_n`, data is written to the RAM.

Data is captured into the memory cell on the low-to-high transition of `wr_n`. If `rd1_addr` port or `rd2_addr` port are the same value as `wr_addr`, `data_in` equals `data_rd1_out` or `data_rd2_out`, respectively, after the low-to-high transition of `wr_n`.

When `cs_n` is high, writing to the RAM is disabled.

## Reset

**rst_n**
This signal is an active-low input that initializes the RAM to zeros if the *rst_mode* parameter is set to 0, independent of the value of `cs_n`. If the *rst_mode* parameter is set to 1, `rst_n` does not affect the RAM, and should be tied high or low. In this case, synthesis optimizes the design, and does not use the `rst_n` signal.

## Making the RAM Scannable

DW_ram_2r_w_a_dff may be made scannable using DFT Compiler. Use the `set_test_hold 1 test_mode` command before `insert_scan`.

The `test_mode` signal, when active high, selects the `test_clk` port to control the capture of data into the RAM. The `test_mode` signal may be tied low if a scannable design is not required. When `test_mode` is tied low, synthesis optimizes the design, and does not connect the `test_mode` and `test_clk` signals to anything.

**Attention** For scannable designs, the `test_mode` signal should only be active during scan shifting (when scan enable is active). When `test_mode` is active, all RAM addresses are written with the `data_in` value at the rising edge of `test_clk`. When `test_mode` and scan enable are both active, the data currently in the RAM are shifted out for viewing the state of the RAM.

## Application Notes

DW_ram_2r_w_a_dff is intended to be used as a small scratch-pad memory or register file. Because DW_ram_2r_w_a_dff is built from the cells within the ASIC cell library, it should be kept small to obtain an efficient implementation. If a larger memory is required, you should consider using a hard macro RAM from the ASIC library in use.

## Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW_SUPPRESS_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

  ```
  `define DW_SUPPRESS_WARN
  ```

- Or, include a command line option to the simulator, such as:

  `+define+DW_SUPPRESS_WARN` (which is used for the Synopsys VCS simulator)

The warning messages for this model include the following:

- If values other than 1 or 0 are present on a clock port, the following message is displayed:

  ```
  WARNING: <instance_path>.<clock_name>_monitor:
      at time = <timestamp>, Detected unknown value, x, on <clock_name> input.
  ```

  To suppress only this warning message for all DWBB components, use the following macro:

  - Define the DW_DISABLE_CLK_MONITOR macro. You can define this macro in the following ways:
    - Specify the Verilog preprocessing macro in Verilog code:

      ```
      `define DW_DISABLE_CLK_MONITOR
      ```

    - Or, include a command line option to the simulator, such as:

      `+define+DW_DISABLE_CLK_MONITOR` (which is used for the Synopsys VCS simulator)

  This message is also suppressed using the DW_SUPPRESS_WARN macro explained earlier.

## Timing Waveforms

The figures in this section show timing diagrams for various conditions of DW_ram_2r_w_a_dff.

**Figure 1-1 Instantiated RAM Timing Waveforms**

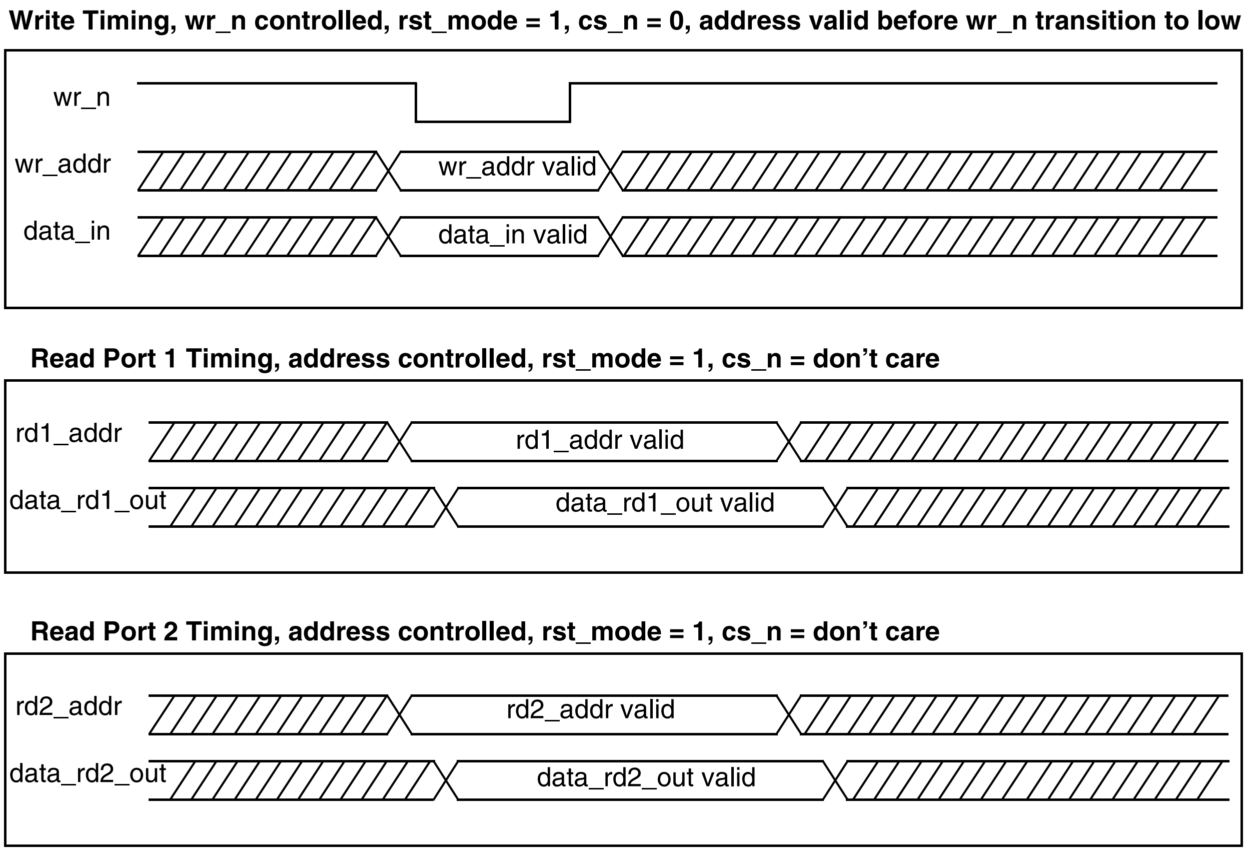
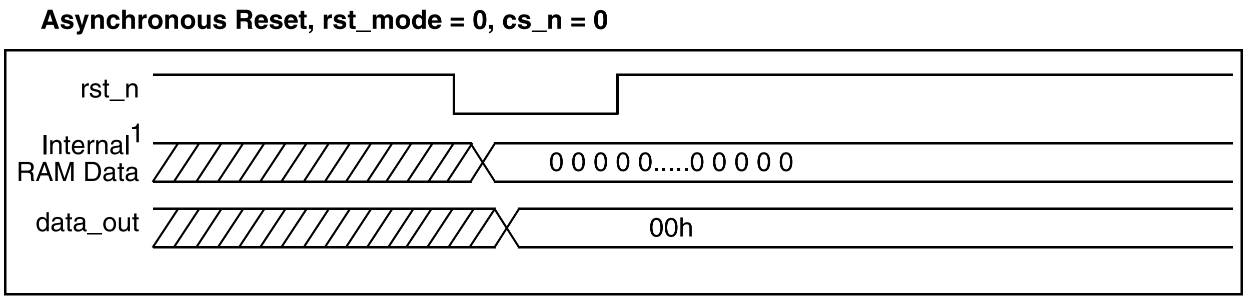

Write Timing, wr_n controlled, rst_mode = 1, cs_n = 0, address valid before wr_n transition to low
wr_n
wr_addr
wr_addr valid
data_in
data_in valid
Read Port 1 Timing, address controlled, rst_mode = 1, cs_n = don’t care
rd1_addr
rd1_addr valid
data_rd1_out
data_rd1_out valid
Read Port 2 Timing, address controlled, rst_mode = 1, cs_n = don’t care
rd2_addr
rd2_addr valid
data_rd2_out
data_rd2_out valid

**Figure 1-2 RAM Reset Timing Waveform**



1 Internal RAM Data is the array of memory bits; the memory is not available to users.

## Related Topics

- Memory – Synchronous RAMs Listing
- DesignWare Building Block IP User Guide

## HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_foundation_comp.all;

entity DW_ram_2r_w_a_dff_inst is
  generic (inst_data_width : INTEGER := 8;
           inst_depth      : INTEGER := 8;
           inst_rst_mode   : INTEGER := 0 );
  port (inst_rst_n     : in std_logic;
        inst_cs_n      : in std_logic;
        inst_wr_n      : in std_logic;
        inst_test_mode : in std_logic;
        inst_test_clk  : in std_logic;
       inst_rd1_addr : in std_logic_vector(bit_width(inst_depth)-1 downto 0);
       inst_rd2_addr : in std_logic_vector(bit_width(inst_depth)-1 downto 0);
        inst_wr_addr : in std_logic_vector(bit_width(inst_depth)-1 downto 0);
        inst_data_in : in std_logic_vector(inst_data_width-1 downto 0);
        data_rd1_out_inst : out std_logic_vector(inst_data_width-1 downto 0);
        data_rd2_out_inst : out std_logic_vector(inst_data_width-1 downto 0)
       );
end DW_ram_2r_w_a_dff_inst;

architecture inst of DW_ram_2r_w_a_dff_inst is
begin

  -- Instance of DW_ram_2r_w_a_dff
  U1 : DW_ram_2r_w_a_dff
    generic map (data_width => inst_data_width,   depth => inst_depth,
                 rst_mode => inst_rst_mode )
    port map (rst_n => inst_rst_n,   cs_n => inst_cs_n,   wr_n => inst_wr_n,
              test_mode => inst_test_mode,   test_clk => inst_test_clk,
              rd1_addr => inst_rd1_addr,   rd2_addr => inst_rd2_addr,
              wr_addr => inst_wr_addr,   data_in => inst_data_in,
              data_rd1_out => data_rd1_out_inst,
              data_rd2_out => data_rd2_out_inst );
end inst;

-- pragma translate_off
configuration DW_ram_2r_w_a_dff_inst_cfg_inst of DW_ram_2r_w_a_dff_inst is
  for inst
  end for; -- inst
end DW_ram_2r_w_a_dff_inst_cfg_inst;
-- pragma translate_on
```

## HDL Usage Through Component Instantiation - Verilog

```
module DW_ram_2r_w_a_dff_inst( inst_rst_n, inst_cs_n, inst_wr_n,
          inst_test_mode, inst_test_clk, inst_rd1_addr, inst_rd2_addr,
          inst_wr_addr, inst_data_in, data_rd1_out_inst, data_rd2_out_inst );
  parameter data_width = 8;
  parameter depth = 8;
  parameter rst_mode = 0;
  `define bit_width_depth 3 // ceil(log2(depth))

  input inst_rst_n;
  input inst_cs_n;
  input inst_wr_n;
  input inst_test_mode;
  input inst_test_clk;
  input [`bit_width_depth-1 : 0] inst_rd1_addr;
  input [`bit_width_depth-1 : 0] inst_rd2_addr;
  input [`bit_width_depth-1 : 0] inst_wr_addr;
  input [data_width-1 : 0] inst_data_in;
  output [data_width-1 : 0] data_rd1_out_inst;
  output [data_width-1 : 0] data_rd2_out_inst;

  // Instance of DW_ram_2r_w_a_dff
  DW_ram_2r_w_a_dff #(data_width,   depth,   rst_mode)
    U1 (.rst_n(inst_rst_n),   .cs_n(inst_cs_n),   .wr_n(inst_wr_n),
        .test_mode(inst_test_mode),   .test_clk(inst_test_clk),
        .rd1_addr(inst_rd1_addr),   .rd2_addr(inst_rd2_addr),
        .wr_addr(inst_wr_addr),   .data_in(inst_data_in),
        .data_rd1_out(data_rd1_out_inst),
        .data_rd2_out(data_rd2_out_inst) );

endmodule
```

## Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

| Date | Release | Updates |
|---|---|---|
| July 2020 | DWBB_201912.5 | ■ Adjusted content and title of "Suppressing Warning Messages During Verilog Simulation" on page 4 and added the DW_SUPPRESS_WARN macro |
| October 2019 | DWBB_201903.5 | ■ Added the "Disabling Clock Monitor Messages" section |
| January 2019 | DWBB_201806.5 | ■ Updated example in "HDL Usage Through Component Instantiation - VHDL" on page 6<br>■ Added this Revision History table and the document links on this page |